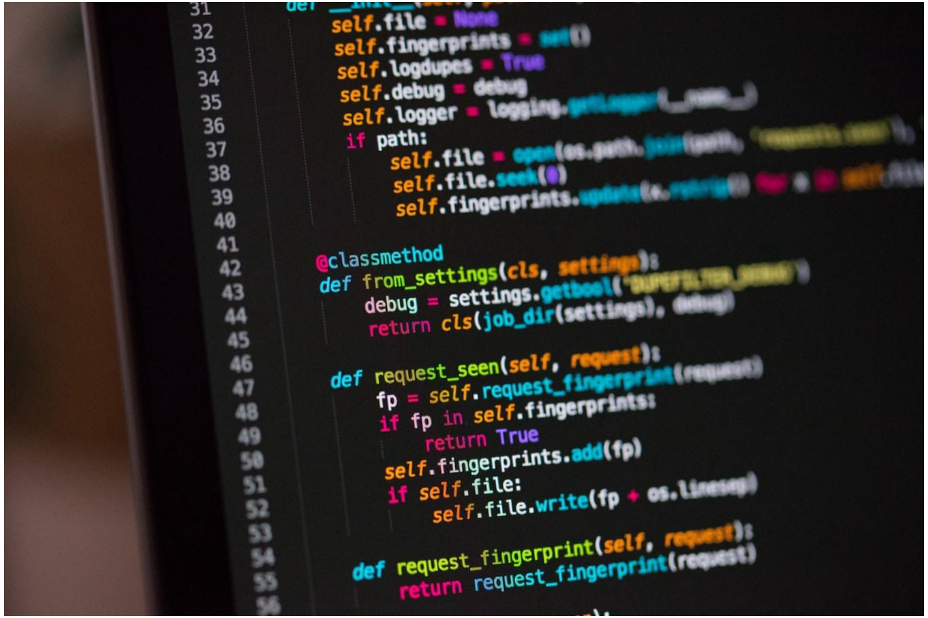
Hack ~~my~~ your code

- Prevent security issues in the code
- Find issues in existing code

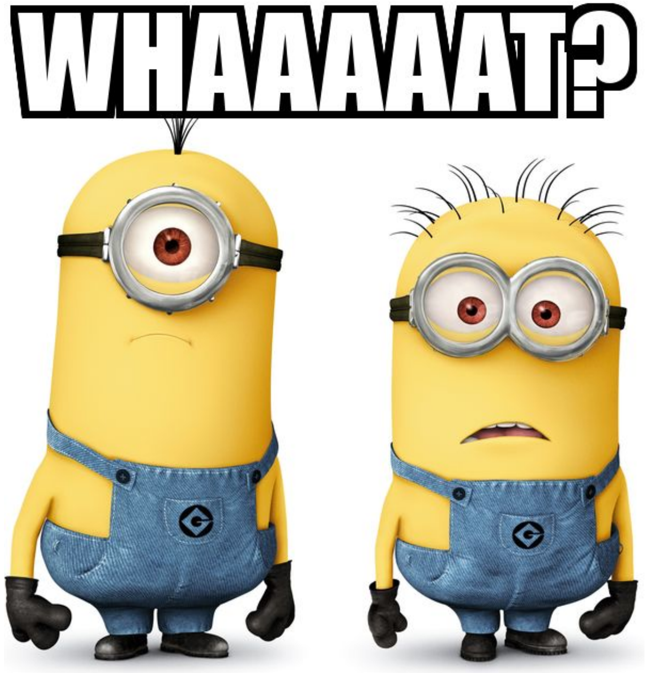
What is secure coding?



Secure coding is the practice of writing software that's protected from vulnerabilities.

Ridiculous excuses of Developers

- This bug is not a “degradation”.
- No one will do this attack!
- We know this bug for many years, **why is it security now?!**
- We know it's the default, but the administrator can turn it off.
- We are not responsible for the security part.



Security Focus in Development Process

Requirements

- Prepare security requirements
- Threat modeling

Design and Development

- Threat modeling updates → Static analysis
- Security design review → Vulnerability scanning
- Code Review

Testing

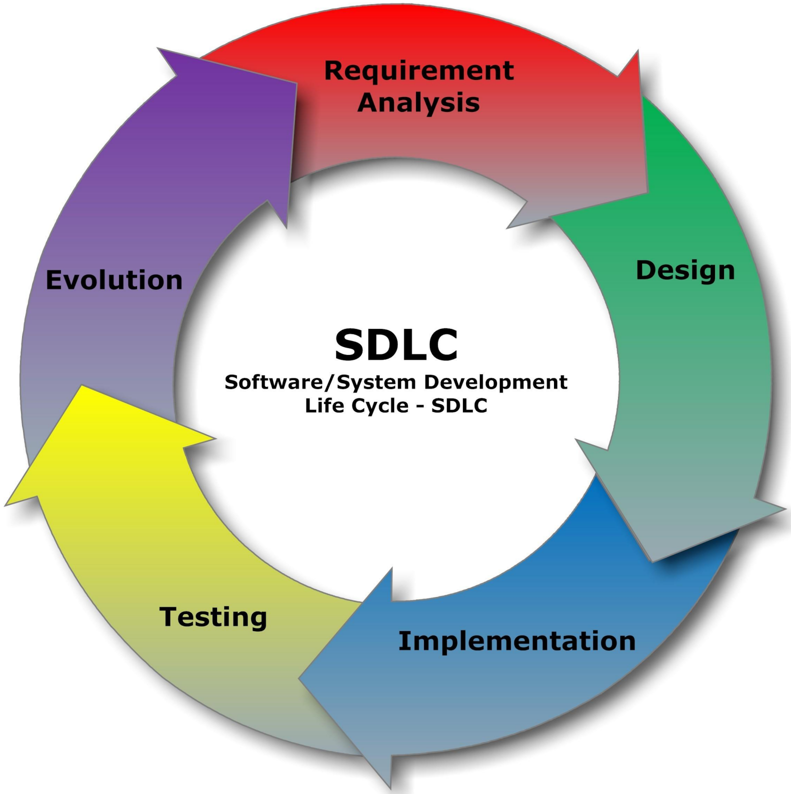
- Dynamic analysis
- Security review
- Third-party Penetration testing

Release

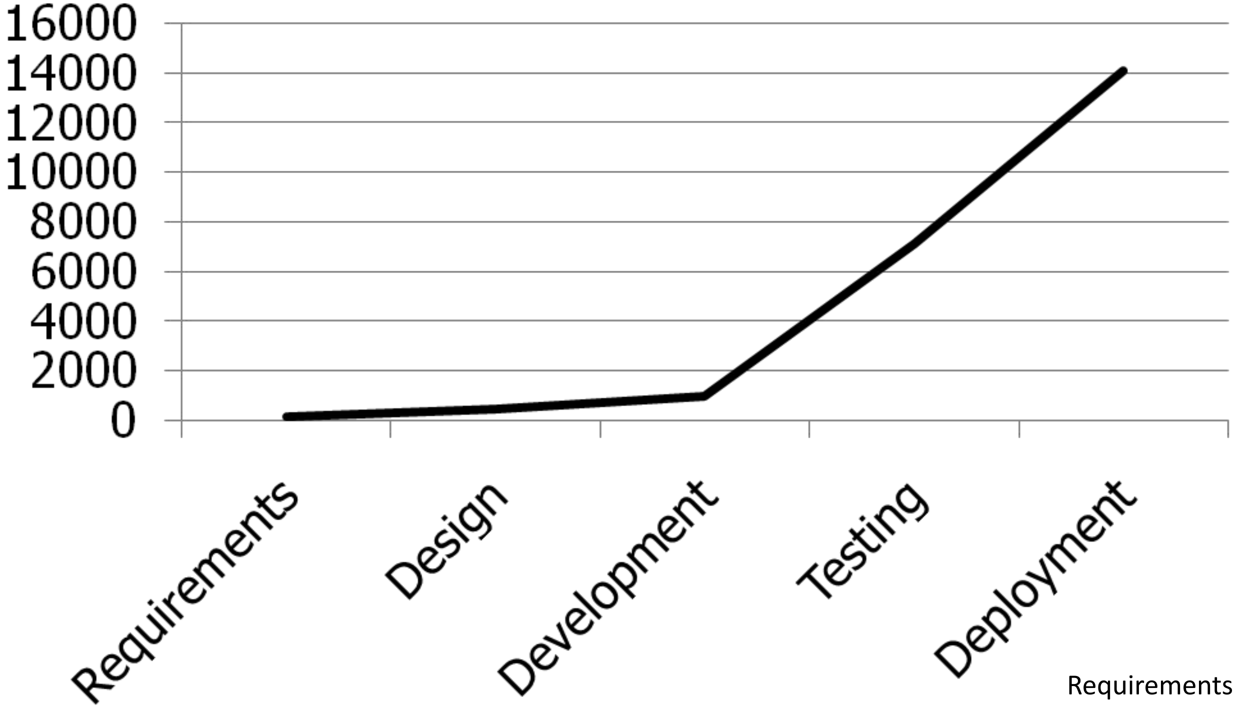
- Final privacy review
- Open-source licensing review

Sustain

- Third-party software tracking and review



Cost Per Defect



Requirements	$139
Design	$455
Development	$977
Testing	$7,136
Deployment	$14,102

Estimates provided by IEEE Computer Society

Defensive programming: the good, the bad and the ugly

- **Defensive programming focuses on error handling**

```
void export (final File file) {
    // export the report to the file
}
```

UGLY

- **What happens if file == NULL?**

BAD

```
void export (final File file) {
  if (file == null) {
    throw new IllegalArgumentException( "file is null; can't export." );
  }
  // export the report to the file
}
```

Defensive programming: the good, the bad and the ugly

- **If the file already exists?**

report will silently overwrite it

GOOD

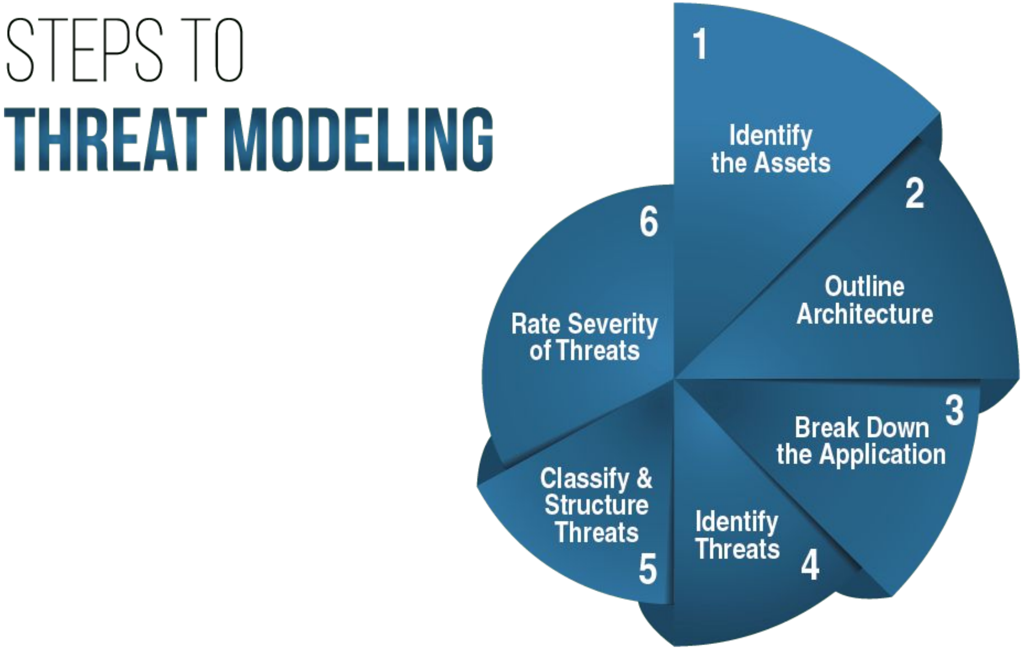
```
void export(final File file) {
  if (file == null) {
    throw new IllegalArgument[illegible]n( "file is null; can't export." );
  }

  if (file.exists()) {
    throw new Illega[illegible]tException( "file already exists." );
  }

  // export th[illegible] to the file
}
```

Defensive programming is not enough

Threat Modeling

"Threat modeling at the design phase is really the only way to bake security into the SDLC." – Michael Howard, Microsoft

…a security control performed during the architecture and design phase of the Software Development Life Cycle (SDLC) to identify and reduce risk within software.

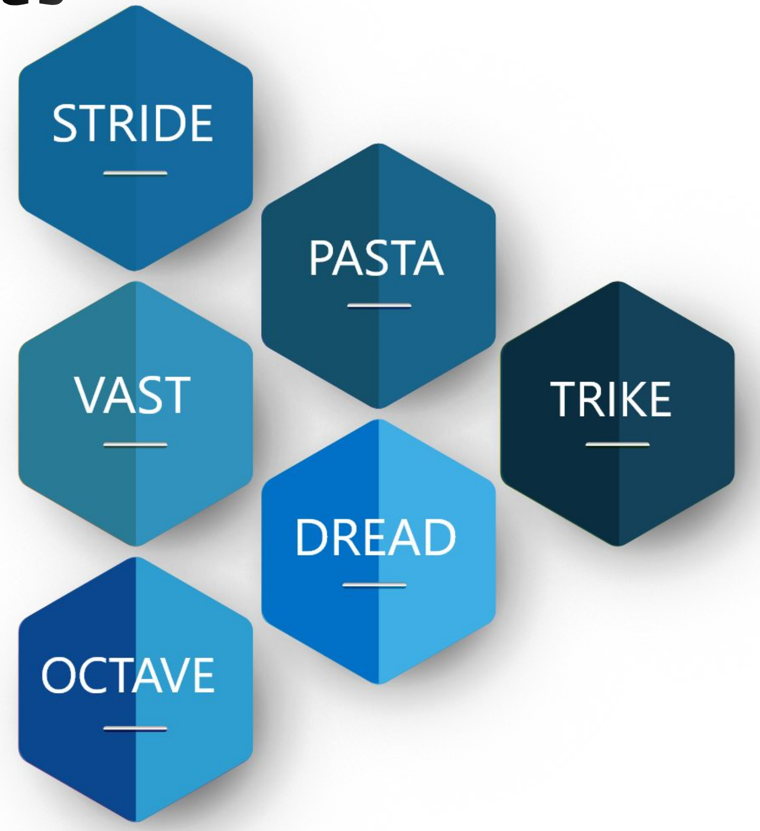


Threat Modeling Approaches

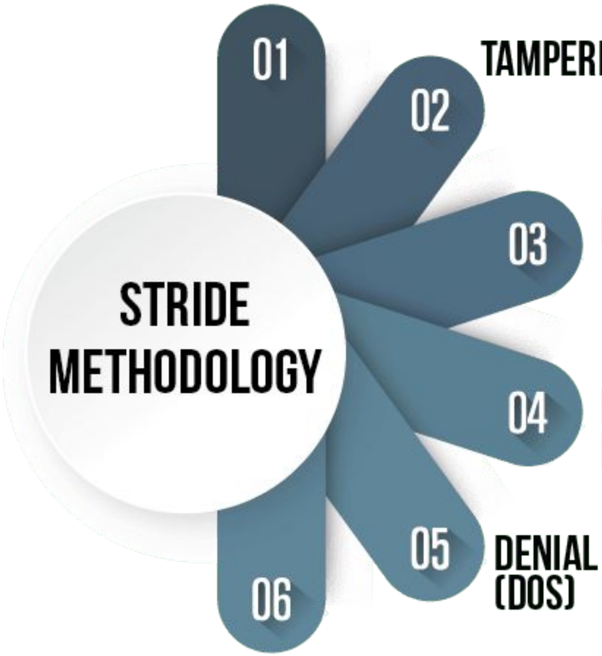
- **Attack Centric**
  - Evaluates from the point of view of an attacker
- **Defense Centric**
  - Evaluates weakness in security controls
- **Asset Centric**
  - Evaluates from asset classification and value
- **Hybrid**
  - Evaluates application design using combination of methodologies to meet security objectives

Threat Modeling Methodologies

https://blog.eccouncil.org/threat-modeling-methodologies-tools-and-processes/



STRIDE Methodology (Developer Focused)



SPOOFING → IP spoofing, user identity spoofing, DNS spoofing, MAC spoofing, MITM

TAMPERING → Change data in transit, forge logs, forge operations, delete logs

REPUDIATION → User denies performing operation

INFORMATION DISCLOSURE
→ Get sensitive information (passwords, encryption keys)
→ Sniffing
→ Reveal system design/architecture

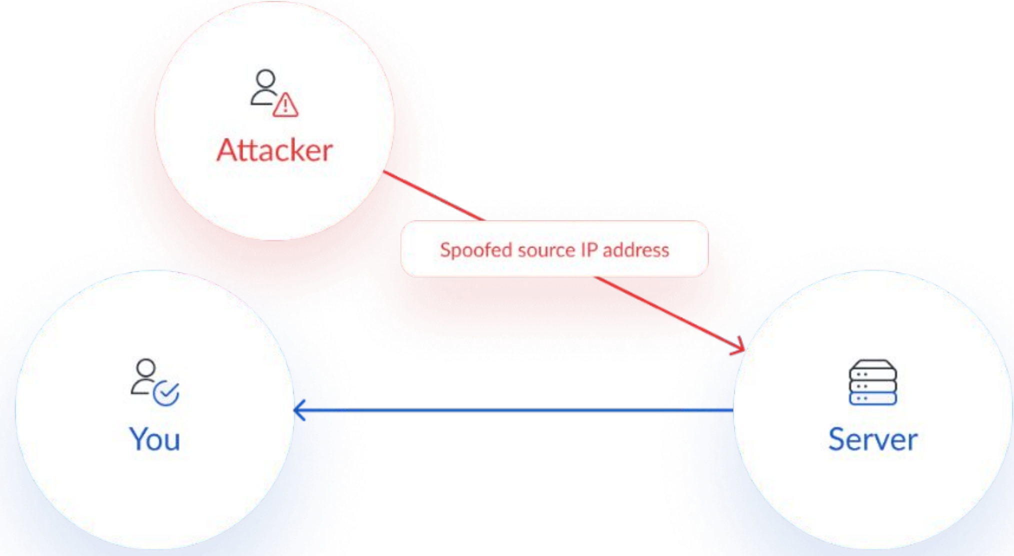
DENIAL OF SERVICE (DOS)
→ Application crash
→ Large memory allocations
→ CPU usage
→ Flooding

ELEVATION OF PRIVILEGE
→ Execute (inject) arbitrary code
→ Get access to restricted resource

IP Spoofing

... is the creation of Internet Protocol (IP) packets which have a modified source address in order to either hide the identity of the sender, to impersonate another computer system, or both.

IPv4 Network Packet Headers



Steps to Avoid IP Spoofing

- deep packet inspection (DPI), which uses granular analysis of all packet headers rather than just source IP address.
- Use secure encryption protocols to secure traffic to and from your server.
- firewall to help protect your network by filtering traffic with spoofed IP addresses, verifying that traffic, and blocking access by unauthorized outsider.
- etc.

OpenSSH S/Key info disclosure

If "**ChallengeResponseAuthentication**" is set to "**Yes**", which is the def[illegible] OpenSSH allows the user to login by using S/KEY in the form of **'ssh userid:skey at hostname'**.

Steps to repr[illegible]

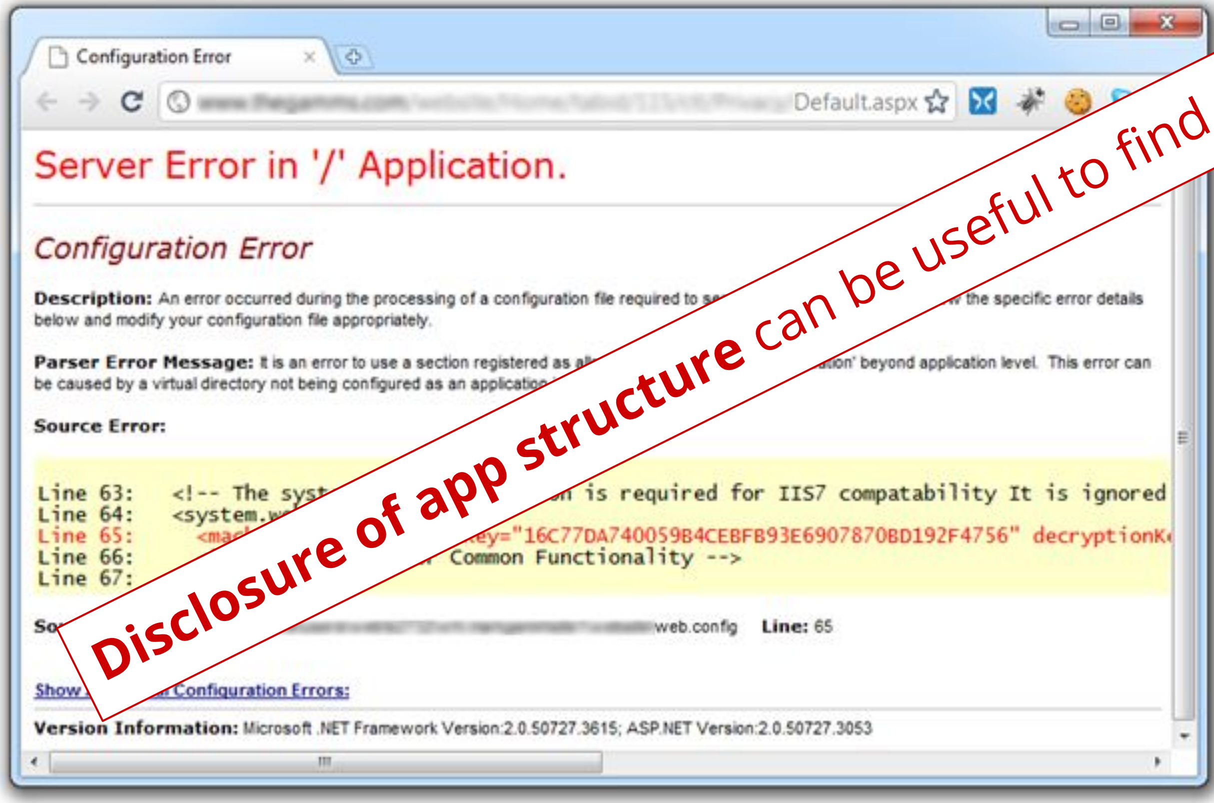
$ ss[illegible]somewhere
[illegible]on denied (publickey,keyboard-interactive).

[illegible] ssh user:skey@somewhere
otp-md5 99 some04578
S/Key Password:
$

Discloses the existence of system accounts.

https://cxsecurity.com/issue/WLB-2007040138

Disclosing info in Error Message



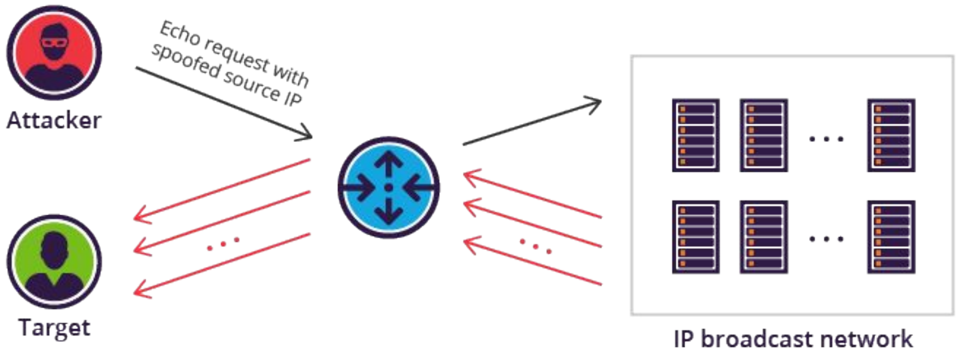
Disclosure of app structure can be useful to find attack vectors.

Impact:

- Confidentiality

DDoS: Smurf Attack

Formula: flood with Internet Control Message Protocol (ICMP) packets + spoofed IP address = Smurf Attack.



A Smurf attack scenario:

1. Smurf malware is used to generate a fake Echo request containing a spoofed source IP, which is actually the target server address.
2. The request is sent to an intermediate IP broadcast network.
3. The request is transmitted to all of the network hosts on the network.
4. Each host sends an ICMP response to the spoofed source address.
5. With enough ICMP responses forwarded, the target server is brought down.

Prevent Smurf Attack

To avoid being the amplifier, you should

- **disable IP-directed broadcast on the router** – this will make it deny the broadcast traffic to the internal network from other networks.

- **apply an outbound filter to your perimeter router**, as well as configuring hosts and routers not to respond to ICMP echo requests.

To avoid being the victim, you should:

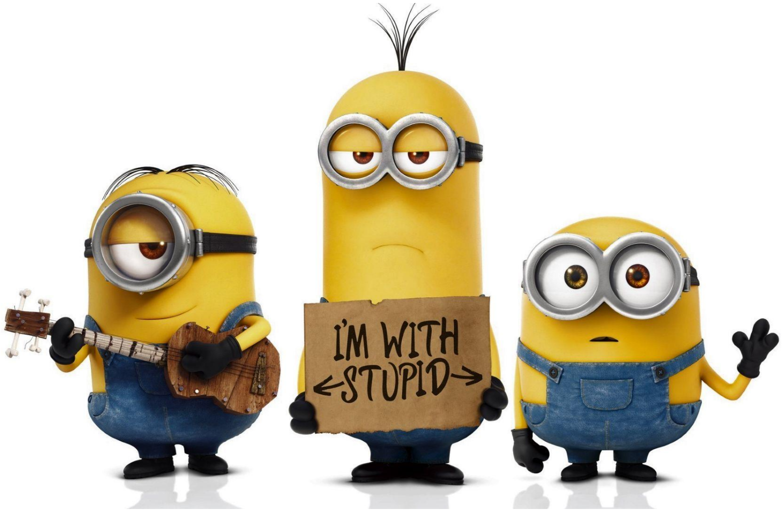
- **firewalls**
- **traffic network monitoring**
- **build redundancy.** Your servers should spread across multiple data centres and have a good load balancing system for traffic distribution. The data centres should be, if possible, in different regions of the same country or even in different countries and should be connected to different networks.

DoS: Algorithmic Complexity Attack

*An **Algorithmic Complexity** (AC) attack is a resource exhaustion attack that takes advantage of worst-case performance in server-side algorithms.*

Developers select algorithms mostly by

- performance
- ease of implementation
- top answer on StackOverflow



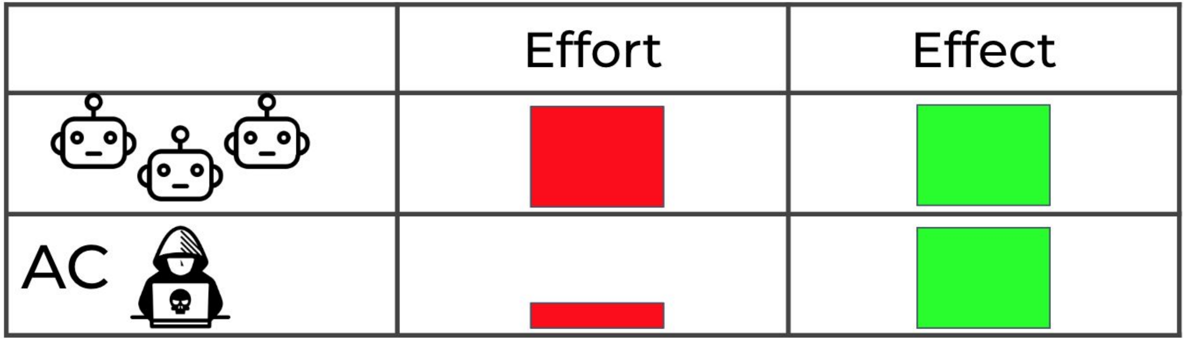
DoS: How do AC vulnerabilities differ from other Attacks

Typical DoS

- dedicate significant resources to the attack
- most commonly use a **botnet** of thousands or millions of nodes → **symmetric effort** on part of the attacker vs. the effect on the target

AC

- typically single user
- small payload → a disproportionately **powerful effect**



DoS: AC Historical Examples

1. Hashtable DoS Attacks: watch

   In 2011, researchers Alexander ‘alech’ Klink and Julian ‘zeri’ Wälde found vulnerabilities in several hash table implementations, including the built-in hash tables in Java, PHP, and Python. These hash tables utilized a linked list for storing hash collisions.

2. Decompression Bombs: watch

   Decompression bombs exploit the ability of efficient compression algorithms to compress a large amount of (typically low entropy) data into a small package. Decompression bombs exist in almost any format that allows compression, from images (png, jpeg) to archives (gz, zip) and other documents (pdf, xml).

3. REDoS: watch

   REDoS, or Regular Expression Denial of Service, refers to a class of vulnerabilities in regular expression parsing engines. If a regular expression allows for “catastrophic backtracking”, then the parser will require processing time that grows exponentially with the candidate string’s length.

DoS: AC Maybe Not So Historical Examples

In **2016**, **StackExchange** experienced a half hour outage due to a **bad regex**.

You can read their post-mortem here.

On **July 2 2019**, **Cloudflare** experienced a blackout due to a **poorly implemented regex**.

You can read their post-mortem here.

DoS: What can be done about AC?

1. **Select a new algorithm.**

   As a result of the hash table collision attacks in 2011, most programming languages changed the data structure, used as bins, for their hash table implementation.

2. **Use input sanitization.**

   Sometimes the AC vulnerability present in a given algorithm only happens for a specific class of inputs. You can restrict the input space a user can submit by placing explicit limits in your application (e.g. limit the length of input, the use of certain options or characters, etc.).

3. **Implement hard resource limits.**

   Many applications will abort decompression when they encounter a decompression bomb by refusing to extract data beyond a certain size.

Vulnerability vs Attack vectors

Vulnerability		Attack vectors
Buffer overflow	→	DoS: process crash Arbitrary code execution
Format string flaws	→	Information disclosure Arbitrary code execution
Integer overflows	→	DoS: large memory allocations Buffer overflow
Race conditions	→	Elevation of privilege
Insecure password management	→	Disclosure of sensitive information
Insecure logging	→	Repudiation
Improper use of SSL	→	Spoofing, impersonation, MITM
Insufficient randomness	→	Session hijacking Information disclosure

PASTA Methodology (Attacker Focused)

→ What is important to your business

→ Create a picture of what you are protecting

→ Map the relationships between components

→ Research the credible threats that affect your business area

→ Map which weaknesses will break under threats

→ Emulate the attacks that could exploit any identified weaknesses

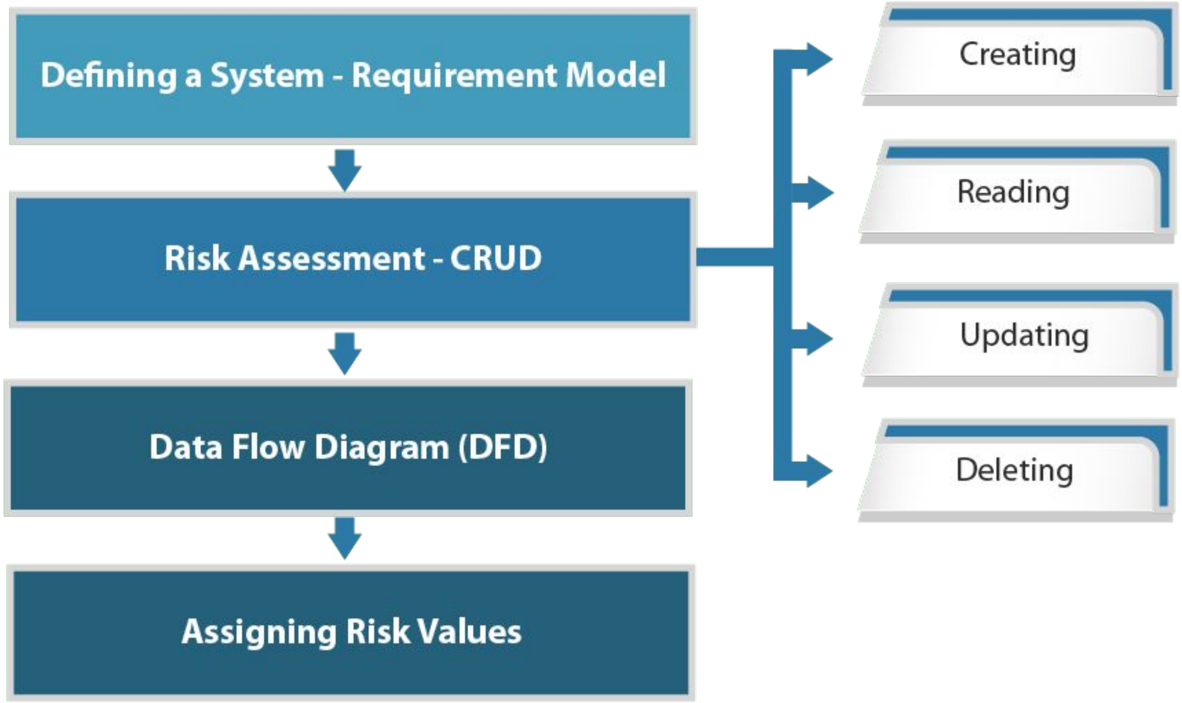
→ This stage uses the answers from earlier stages

PASTA Methodology: Benefits

- Put security at the centre of the entire business.
- Get a full picture of the threats an organisation may face.
- Understanding of the evolving cyber threat landscape.
- Informed decision making.

TRIKE Methodology (Acceptable Risks Focused)

Trike is a unified methodology for carrying out security threat modeling. This is accomplished through the generation of threat models from a risk management perspective.



TRIKE Methodology: Goals

- Ensure that the risk this system entails to each asset is acceptable to all stakeholders.
- Be able to tell whether we have done this.
- Communicate what we’ve done and its effects to the stakeholders.
- Empower stakeholders to understand and reduce the risks to them and other stakeholders implied by their actions within their domains.

DREAD Methodology (Risks Assessment Focused)

Creates Rating System

DREAD: Damage Potential

Rating	Damage
0	No damage
5	Information disclosure
8	Individual/employer non-sensitive user data compromised
9	Administrative non-sensitive data compromised
10	Information system or data destruction or application unavailability

DREAD: Reproducible

Rating	How easy is it to reproduce the attack?
0	Difficult or Impossible
5	Complex
7.5	Easy for authenticated user
10	Very easy through web browser, no authentication

DREAD: Exploitability

Rating	What is required to exploit this threat?
2.5	Advanced programming and networking skills
5	Using available attack tools
9	A web application proxy tool
10	Web browser

DREAD: Affected users

Rating	How many users affected?
0	No users affected
2.5	Individual user
6	Few users
8	Administrative users
10	All users

DREAD: Discoverability

Rating	How easy is it to discover the threat?
0	Very hard
5	Can figure it out by HTTP requests
8	Already in the public domain and can easily be discovered
10	Visible in the web browser address bar or in a form

DREAD: Final Decision

Risk Rating	**DREAD Score**	**Comments**
Critical	40-50	Critical vulnerability, should be considered immediately for review and resolution
High	25-39	Severe vulnerability, should be considered for review and resolution within a short period of time
Medium	11-24	Moderate risk finding or vulnerabilities should be considered once severe and critical risks have been addressed
Low	1-10	Low risk and does not pose significant risk to the IT infrastructure

VAST Methodology

Automation

- Eliminates Repetition in Threat Modeling
- Ongoing Threat Modeling
- Scaled to Encompass the Entire Enterprise

Integration

- Integration with Tools Throughout the SDLC
- Supports the Agile DevOps

Collaboration

Key Stakeholders Collaboration: App Developers, Systems Architects, Security Team, and Senior Executives

VISUAL

AGILE

SIMPLE **T**REAT

Goal: Scale threat modeling solution

Developers + DevOPS + SecOPS

Threat Modeling Methodologies in SDLC

1. What are we building?
   - TRIKE
2. What can go wrong?
   - PASTA
   - STRIDE
   - Kill Chains
   - CAPEC
   - VAST
3. What are we going to do about that?
   - Estimate risks: DREAD
   - Fix the critical issues
4. Did we do a good enough job?
   - Final test

MAIN GOAL: The different methodologies may be used on the different levels.

Threat Modeling

- **Improves Security**
  - Champions threat analysis
  - Uncovers logical/architectural vulnerabilities
  - Reduces risk and minimizes impact
- **Drives Testing**
  - Validates design meets security requirements
  - Reduces scope of code inspection
  - Serves as a guide for verification testing
- **Reduces Cost**
  - Identifies expensive mistakes early on
  - Improve understanding and structure of application
  - Decreases new hire ramp up time

Decide when enough **is enough**